

## 16 Maximum Satisfiability

The maximum satisfiability problem has been a classical problem in approximation algorithms. More recently, its study has led to crucial insights in the area of hardness of approximation (see Chapter 29). In this chapter, we will use LP-rounding, with randomization, to obtain a  $3/4$  factor approximation algorithm. We will derandomize this algorithm using the *method of conditional expectation*.

**Problem 16.1 (Maximum satisfiability (MAX-SAT))** Given a conjunctive normal form formula  $f$  on Boolean variables  $x_1, \dots, x_n$ , and non-negative weights,  $w_c$ , for each clause  $c$  of  $f$ , find a truth assignment to the Boolean variables that maximizes the total weight of satisfied clauses. Let  $\mathcal{C}$  represent the set of clauses of  $f$ , i.e.,  $f = \bigwedge_{c \in \mathcal{C}} c$ . Each clause is a disjunction of literals; each literal being either a Boolean variable or its negation. Let  $\text{size}(c)$  denote the *size* of clause  $c$ , i.e., the number of literals in it. We will assume that the sizes of clauses in  $f$  are arbitrary.

For any positive integer  $k$ , we will denote by MAX- $k$ SAT the restriction of MAX-SAT to instances in which each clause is of size at most  $k$ . MAX-SAT is **NP-hard**; in fact, even MAX-2SAT is **NP-hard** (in contrast, 2SAT is in **P**). We will first present two approximation algorithms for MAX-SAT, having guarantees of  $1/2$  and  $1 - 1/e$ , respectively. The first performs better if the clause sizes are large, and the second performs better if they are small. We will then show how an appropriate combination of the two algorithms achieves the promised approximation guarantee.

In the interest of minimizing notation, let us introduce common terminology for all three algorithms. Random variable  $W$  will denote the total weight of satisfied clauses. For each clause  $c$ , random variable  $W_c$  denotes the weight contributed by clause  $c$  to  $W$ . Thus,  $W = \sum_{c \in \mathcal{C}} W_c$  and

$$\mathbf{E}[W_c] = w_c \cdot \mathbf{Pr}[c \text{ is satisfied}].$$

(Strictly speaking, this is abuse of notation, since the randomization used by the three algorithms is different.)

## 16.1 Dealing with large clauses

The first algorithm is straightforward. Set each Boolean variable to be True independently with probability  $1/2$  and output the resulting truth assignment, say  $\tau$ . For  $k \geq 1$ , define  $\alpha_k = 1 - 2^{-k}$ .

**Lemma 16.2** *If  $\text{size}(c) = k$ , then  $\mathbf{E}[W_c] = \alpha_k w_c$ .*

**Proof:** Clause  $c$  is not satisfied by  $\tau$  iff all its literals are set to False. The probability of this event is  $2^{-k}$ .  $\square$

For  $k \geq 1$ ,  $\alpha_k \geq 1/2$ . By linearity of expectation,

$$\mathbf{E}[W] = \sum_{c \in \mathcal{C}} \mathbf{E}[W_c] \geq \frac{1}{2} \sum_{c \in \mathcal{C}} w_c \geq \frac{1}{2} \text{OPT},$$

where we have used a trivial upper bound on OPT — the total weight of clauses in  $\mathcal{C}$ .

Instead of converting this into a high probability statement, with a corresponding loss in guarantee, we show how to derandomize this procedure. The resulting algorithm deterministically computes a truth assignment such that the weight of satisfied clauses is  $\geq \mathbf{E}[W] \geq \text{OPT}/2$ .

Observe that  $\alpha_k$  increases with  $k$  and the guarantee of this algorithm is  $3/4$  if each clause has two or more literals. (The next algorithm is designed to deal with unit clauses more effectively.)

## 16.2 Derandomizing via the method of conditional expectation

We will critically use the self-reducibility of SAT (see Section A.5). Consider the self-reducibility tree  $T$  for formula  $f$ . Each internal node at level  $i$  corresponds to a setting for Boolean variables  $x_1, \dots, x_i$ , and each leaf represents a complete truth assignment to the  $n$  variables. Let us label each node of  $T$  with its conditional expectation as follows. Let  $a_1, \dots, a_i$  be a truth assignment to  $x_1, \dots, x_i$ . The node corresponding to this assignment will be labeled with  $\mathbf{E}[W | x_1 = a_1, \dots, x_i = a_i]$ . If  $i = n$ , this is a leaf node and its conditional expectation is simply the total weight of clauses satisfied by its truth assignment.

**Lemma 16.3** *The conditional expectation of any node in  $T$  can be computed in polynomial time.*

**Proof:** Consider a node  $x_1 = a_1, \dots, x_i = a_i$ . Let  $\phi$  be the Boolean formula, on variables  $x_{i+1}, \dots, x_n$ , obtained for this node via self-reducibility. Clearly,

the expected weight of satisfied clauses of  $\phi$  under a random truth assignment to the variables  $x_{i+1}, \dots, x_n$  can be computed in polynomial time. Adding to this the total weight of clauses of  $f$  already satisfied by the partial assignment  $x_1 = a_1, \dots, x_i = a_i$  gives the answer.  $\square$

**Theorem 16.4** *We can compute, in polynomial time, a path from the root to a leaf such that the conditional expectation of each node on this path is  $\geq \mathbf{E}[W]$ .*

**Proof:** The conditional expectation of a node is the average of the conditional expectations of its two children, i.e.,

$$\mathbf{E}[W|x_1 = a_1, \dots, x_i = a_i] = \mathbf{E}[W|x_1 = a_1, \dots, x_i = a_i, x_{i+1} = \text{True}]/2 + \mathbf{E}[W|x_1 = a_1, \dots, x_i = a_i, x_{i+1} = \text{False}]/2.$$

The reason, of course, is that  $x_{i+1}$  is equally likely to be set to True or False. As a result, the child with the larger value has a conditional expectation at least as large as that of the parent. This establishes the existence of the desired path. As a consequence of Lemma 16.3, it can be computed in polynomial time.  $\square$

The deterministic algorithm follows as a corollary of Theorem 16.4. We simply output the truth assignment on the leaf node of the path computed. The total weight of clauses satisfied by it is  $\geq \mathbf{E}[W]$ .

Let us show that the technique outlined above can, in principle, be used to derandomize more complex randomized algorithms. Suppose the algorithm does not set the Boolean variables independently of each other (for instance, see Remark 16.6). Now,

$$\begin{aligned} \mathbf{E}[W|x_1 = a_1, \dots, x_i = a_i] = & \\ \mathbf{E}[W|x_1 = a_1, \dots, x_i = a_i, x_{i+1} = \text{True}] \cdot \Pr[x_{i+1} = \text{True}|x_1 = a_1, \dots, x_i = a_i] + & \\ \mathbf{E}[W|x_1 = a_1, \dots, x_i = a_i, x_{i+1} = \text{False}] \cdot \Pr[x_{i+1} = \text{False}|x_1 = a_1, \dots, x_i = a_i]. & \end{aligned}$$

The sum of the two conditional probabilities is again 1, since the two events are exhaustive. So, the conditional expectation of the parent is still a convex combination of the conditional expectations of the two children. If we can determine, in polynomial time, which of the two children has a larger value, we can again derandomize the algorithm. However, computing the conditional expectations may not be easy. Observe how critically independence was used in the proof of Lemma 16.3. It was because of independence that we could assume a random truth assignment on Boolean variables  $x_{i+1}, \dots, x_n$  and thereby compute the expected weight of satisfied clauses of  $\phi$ .

In general, a randomized algorithm may pick from a larger set of choices and not necessarily with equal probability. But once again a convex combination of the conditional expectations of these choices, given by the probabilities

of picking them, equals the conditional expectation of the parent. Hence there must be a choice that has at least as large a conditional expectation as the parent.